redis

A persistent key-value database with built-in net interface written in ANSI-C for Posix systems

ProtocolSpecification

Updated Sep 14, 2010 by soveran

Protocol Specification

The Redis protocol is a compromise between the following things:

- Simple to implement.
- Fast to parse by a computer.
- Easy enough to parse by a human.

Networking layer

A client connects to a Redis server creating a TCP connection to the port 6379. Every Redis command or data transmitted by the client and the server is terminated by "\r\n" (CRLF).

Requests

Redis accepts commands composed of different arguments. Once a command is received, it is processed and a reply is sent back to the client.

The new unified request protocol

The new unified protocol was introduced in Redis 1.2, but it became the standard way for talking with the Redis server in Redis 2.0.

In the unified protocol all the arguments sent to the Redis server are binary safe. This is the general form:

```
*<number of arguments> CR LF
$<number of bytes of argument 1> CR LF
<argument data> CR LF
...
$<number of bytes of argument N> CR LF
<argument data> CR LF
```

See the following example:

```
*3
$3
SET
$5
mykey
$7
myvalue
```

This is how the above command looks as a quoted string, so that it is possible to see the exact value of every byte in the query:

```
"*3\r\n$3\r\nSET\r\n$5\r\nmykey\r\n$8\r\nmyvalue\r\n"
```

As you will see in a moment this format is also used in Redis replies. The format used for every argument "$6\r\nmydata\r\n" is called a Bulk Reply. While the actual unified request protocol is what Redis uses to return list of items, and is called a Multi Bulk Reply. It is just the sum of N different Bulk Replies prefixed by a `*<argc>\r\n` string where `<argc>` is the number of arguments (Bulk Replies) that will follow.

Replies

Redis will reply to commands with different kinds of replies. It is possible to check the kind of reply from the first byte sent by the server:

- With a single line reply the first byte of the reply will be "+"
- With an error message the first byte of the reply will be "-"
- With an integer number the first byte of the reply will be ":"
- With bulk reply the first byte of the reply will be "$"
- With multi-bulk reply the first byte of the reply will be "*"

Single line reply

A single line reply is in the form of a single line string starting with "+" terminated by "\r\n". For example:

```
+OK
```

The client library should return everything after the "+", that is, the string "OK" in the example.

The following commands reply with a single line reply: PING, SET, SELECT, SAVE, BGSAVE, SHUTDOWN, RENAME, LPUSH, RPUSH, LSET, LTRIM

Error reply

Errors are sent exactly like Single Line Replies. The only difference is that the first byte is "-" instead of "+".

Error replies are only sent when something strange happened, for instance if you try to perform an operation against the wrong data type, or if the command does not exist and so forth. So an exception should be raised by the library client when an Error Reply is received.

Integer reply

This type of reply is just a CRLF terminated string representing an integer, prefixed by a ":" byte. For example ":0\r\n", or ":1000\r\n" are integer replies.

With commands like INCR or LASTSAVE using the integer reply to actually return a value there is no special meaning for the returned integer. It is just an incremental number for INCR, a UNIX time for LASTSAVE and so on.

Some commands like EXISTS will return 1 for true and 0 for false.

Other commands like SADD, SREM and SETNX will return 1 if the operation was actually done, 0 otherwise.

The following commands will reply with an integer reply: SETNX, DEL, EXISTS, INCR, INCRBY, DECR, DECRBY, DBSIZE, LASTSAVE, RENAMENX, MOVE, LLEN, SADD, SREM, SISMEMBER, SCARD

Bulk replies

Bulk replies are used by the server in order to return a single binary safe string.

```
C: GET mykey
S: $6
S: foobar
```

The server sends as the first line a "$" byte followed by the number of bytes of the actual reply, followed by CRLF, then the actual data bytes are sent, followed by additional two bytes for the final CRLF. The exact sequence sent by the server is:

```
"$6\r\nfoobar\r\n"
```

If the requested value does not exist the bulk reply will use the special value -1 as data length, example:

```
C: GET nonexistingkey
S: $-1
```

The client library API should not return an empty string, but a nil object, when the requested object does not exist. For example a Ruby library should return 'nil' while a C library should return NULL (or set a special flag in the reply object), and so forth.

Multi-Bulk replies

Commands like LRANGE need to return multiple values (every element of the list is a value, and LRANGE needs to return more than a single element). This is accomplished using multiple bulk writes, prefixed by an initial line indicating how many bulk writes will follow. The first byte of a multi bulk reply is always `*`. Example:

```
C: LRANGE mylist 0 3
S: *4
S: $3
S: foo
S: $3
S: bar
S: $5
S: Hello
S: $5
S: World
```

As you can see the multi bulk reply is exactly the same format used in order to send commands to the Redis server unsing the unified protocol.

The first line the server sent is **"4\r\n" in order to specify that four bulk replies will follow. Then every bulk write is transmitted.**

If the specified key does not exist, instead of the number of elements in the list the special value -1 is sent as count. Example:

```
C: LRANGE nokey 0 1
S: *-1
```

A client library API SHOULD return a nil object and not an empty list when this happens. This makes possible to distinguish between empty list and other error conditions (for instance a timeout condition in the BLPOP command).

Nil elements in Multi-Bulk replies

Single elements of a multi bulk reply may have -1 length, in order to signal that this elements are missing and not empty strings. This can happen with the SORT command when used with the GET *pattern* option when the specified key is missing. Example of a multi bulk reply containing an empty element:

```
S: *3
S: $3
S: foo
S: $-1
S: $3
S: bar
```

The second element is nul. The client library should return something like this:

```
["foo",nil,"bar"]
```

Multiple commands and pipelining

A client can use the same connection in order to issue multiple commands. Pipelining is supported so multiple commands can be sent with a single write operation by the client, it is not needed to read the server reply in order to issue the next command. All the replies can be read at the end.

Usually Redis server and client will have a very fast link so this is not very important to support this feature in a client implementation, still if an application needs to issue a very large number of commands in short time to use pipelining can be much faster.

The old protocol for sending commands

Before of the Unified Request Protocol Redis used a different protocol to send commands, that is still supported since it is simpler to type by hand via telnet. In this protocol there are two kind of commands:

- Inline commands: simple commands where argumnets are just space separated strings. No binary safeness is possible.
- Bulk commands: bulk commands are exactly like inline commands, but the last argument is handled in a special way in order to allow for a binary-safe last argument.

Inline Commands

The simplest way to send Redis a command is via **Inline Commands**. The following is an example of a server/client chat using an inline command (the server chat starts with S:, the client chat with C:)

```
C: PING
S: +PONG
```

The following is another example of an INLINE command returning an integer:

```
C: EXISTS somekey
S: :0
```

Since 'somekey' does not exist the server returned ':0'.

Note that the EXISTS command takes one argument. Arguments are separated by spaces.

Bulk commands

Some commands when sent as inline commands require a special form in order to support a binary safe last argument. This commands will use the last argument for a "byte count", then the bulk data is sent (that can be binary safe since the server knows how many bytes to read).

See for instance the following example:

```
C: SET mykey 6
C: foobar
S: +OK
```

The last argument of the commnad is '6'. This specify the number of DATA bytes that will follow, that is, the string "foobar". Note that even this bytes are terminated by two additional bytes of CRLF.

All the bulk commands are in this exact form: instead of the last argument the number of bytes that will follow is specified, followed by the bytes composing the argument itself, and CRLF. In order to be more clear for the programmer this is the string sent by the client in the above sample:

"SET mykey 6\r\nfoobar\r\n"

Redis has an internal list of what command is inline and what command is bulk, so you have to send this commands accordingly. It is strongly suggested to use the new Unified Request Protocol instead.

Comment by braunster, Sep 08, 2009

Under "Bulk Commands" section: it is unclear how the last argument is parsed. What happens to characters following the number of bytes argument? In particular the CRLF pair and any non-numeric characters. What happens if there are characters following the number specified and the terminal CRLF?

Comment by braunster, Sep 08, 2009

Under "Bulk Commands" section: change **"This specify the number of DATA bytes that will follow (note that even this bytes are terminated by two additional bytes of CRLF)." to** "This specifies the number of DATA bytes that will follow. Note: a CRLF must follow the data bytes."

Comment by braunster, Sep 08, 2009

Under "Bulk Replies" section: The comment about what a library should or should not return is not appropriate for a protocol specification. The protocol specification should describe the form of the replies. Return values from a library routing is an implementation issue.

An argument can be made that a bulk reply of "-<something>" should be returned if the item(s) were not found. However, errors don't seem to be well documented here.

Comment by hauptadler, Dec 29, 2009

Pipelineing

If the command from the client doesn't include its own length, nothing provides that the whole command will be read by a single read() on the server side.

Comment by hauptadler, Dec 29, 2009

Ok, understood. Pipelining is only a performance booster, commands from other clients can be inserted from the server's point of view!

Comment by MCSseifer, Feb 04, 2010

This protocol not useful, because per line reading can be very long. Also sets is very necessary thing, but on 1000 elements it's very slowly. Thank for you work.

Comment by laylward, Mar 08, 2010

What happens if the length for a multi-bulk reply is 0. I assume the client library should not need to read anything?

Comment by Gandalf.Software, Mar 25, 2010

This protocol doesn't seem to address at all issues of binary vs. text data or character sets... what if I want to send/return large Unicode strings, for example... or is the protocol just limited to what the backend database supports?

Comment by mihai.bazon, Jun 26, 2010

Can MULTI-BULK result start with:

```
*-1
```

I noticed that currently if I request data from a list that doesn't exist, i.e. "LRANGE inexistent 0 3" I get:

```
*0
```

A direct client implementation would translate this into an empty array. I was wondering if it wouldn't be better to return -1 instead, so that clients can return NULL. Not sure existing clients are prepared for this though, but mine is. :-)

Comment by Bluemangroupie, Aug 05, 2010

The Multi bulk commands section has an example that says that 8 bytes will follow, but 'myvalue' only has 7 bytes.

Comment by shianli.88, Sep 06, 2010

Connecting to 192.168.0.20:6379... Connection established. Escape character is '^@]'. hmset redis_dic aa aa bb bb cc cc

+OK hmget redis_dic aa bb cc

3 $2 aa $2 bb $-1

why?

Comment by project member antirez, Sep 06, 2010

shianli: because HMSET is a bulk command. The best thing is using the new protocol btw:

```
*4
$5
HMSET
$2
aa
$2
bb
$2
cc
```

Comment by shianli.88, Sep 06, 2010

Connecting to 192.168.0.20:6379... Connection established. Escape character is '^@]'. flushall +OK

4 $5 hmset $3 dic $2 aa $2 bb +OK keys 1 $3 dic hkeys dic 1 $2 aa antirez:This protocol can not , because the last set of data lost

Comment by shianli.88, Sep 06, 2010

Connecting to 192.168.0.20:6379... Connection established. Escape character is '^@]'. keys 0 lpush redis_list_aa aa

:1 lpush redis_list_aa bb

:2 lpush redis_list_aa cc

:3 keys 1 $13 redis_list_aa lindex redis_list_aa 1 $0 why?

Comment by project member antirez, Sep 07, 2010

@shianli.88 sorry your problems are trivially fixable carefully reading this page, obviously you need a byte count as last argument for a bulk command. Please check this page and a simple implementation of a Redis client to gain more information about how to talk the wire protocol.

Regards, Salvatore

Enter a comment:

Submit

Wiki markup help hide

```
=Heading1=
==Heading2==
===Heading3===

*bold*    _italic_
`inline code`
escape: `*`

Indent lists 2 spaces:
  * bullet item
  # numbered list

{{{
verbatim code block
}}}

Horizontal rule
----
```

WikiWordLink
[http://domain/page label]
http://domain/page

|| table || cells ||

More examples